MASSE: Modular Automated Syntactic Signature Extraction

Fabrizio Biondi, François Déchelle, Axel Legay

MASSE: Modular Automated Syntactic Signature Extraction

Fabrizio Biondi
CentraleSupélec Rennes, France
Email: fabrizio.biondi@inria.fr

François Dechelle
Teclib', France
Email: fdechelle@teclib.com

Axel Legay
Inria, France
Email: axel.legay@inria.fr

***Abstract*—We present the MASSE architecture, a YARA-based open source client-server malware detection platform. MASSE includes highly effective automated syntactic malware detection rule generation for the clients based on a server-side modular malware detection system. Multiple techniques are used to make MASSE effective at detecting malware while keeping it from disrupting users and hindering reverse-engineering of its malware analysis by malware creators.**

I. Context

Effective malware detection is an important requirement to guarantee system safety and user protection. Malware detection techniques based on syntactic signatures (or "rules") [6], [8] are commonly used in antivirus since their low computational cost allows them to be used on scan the files handled by the system without excessively slowing down the system.

The open-source YARA tool [2] has recently gained popularity due to its efficience, simple rule format, and variety of implemented techniques. In fact, the YARA rule format has become the de facto standard for sharing malware signatures [5]. However, it has not yet been integrated in a complete antivirus architecture including sample analysis and fingerprinting, and the rules used by it are commonly generated by hand or by simple string subtraction techniques [3], [4].

The Modular Automated Syntactic Signature Extraction (MASSE) architecture is a new integrated open source client-server architecture for syntactic malware detection and analysis based on the YARA, developed with Teclib'. MASSE integrates YARA in a distributed system able to detect malware on endpoint systems using YARA, analyze malware with multiple analysis techniques, automatically generate syntactic malware detection rules, and deploy the new rules to the endpoints. The MASSE architecture is freely available to companies and institutions as a complete, modular, self-maintained antivirus solution. Using MASSE, a security department can immediately update the rule database of the whole company, stopping an infection on its tracks and preventing future ones.

II. Objective: a Smart, Modular, Free Antivirus

The goal of MASSE is to create a highly automated malware detection and analysis client-server architecture. No solution controlled by third parties is required, allowing the user to keep confidentiality on their files. Also, full control on the system allows the user to react immediately in case of attack.

a) Architecture: The MASSE architecture is composed of a client side and a server side where the server executes the computationally-heavy malware analysis procedures in an easily extendable and modular way, and based on this analysis generates syntactic rules for lightweight malware detection in the clients. The client performs YARA-based on-access and on-demand malware analysis on the client's system, without consuming too much of the client system's resources or interfering on its normal usage. The server analyzes new samples to detect malware using a modular system based on multiple malware analysis modules. This produces the YARA detection rules for the new malware, and pushes them to the client's rule database. The architecture is depicted in Fig. 1.

The MASSE source code is publicly available, not just allowing for code analysis and auditing but also creating a community of users and developers sharing improvements to the system. The goal of MASSE is to create a community of users improving the main code and implementing and exchanging new malware modules and rule databases, just like a similar community formed around the YARA tool. Still, the architecture must hinder attempts by malware creators to reverse-engineer the malware analysis to protect their malware against it.

b) Modular Malware Analysis: Malware analysis and rule generation are executed in a modular system with public APIs, making them easily extensible and adaptable to the users' needs. Modules can use machine learning, static and dynamic analysis, heuristic techniques and their own rule or signature databases to perform malware analysis and provide a classification result and possibly a confidence level. Also, each module must implement logic to transform the information it used to detect the malware in a fragment of a syntactic rule that can be used by YARA.

III. Proposed Solution

This section presents the solution to achieve the goals described in Section II.

a) Architecture: To guarantee that the architecture does not interfere with the users, malware detection systems have to have a very low positive rate, usually below 0.1% (and 0% on system files) to prevent raising false alarms that would waste the user's time and corrode their trust in the system. MASSE's rule generation uses multiple techniques to reduce false positives. First of all, the server maintains a large

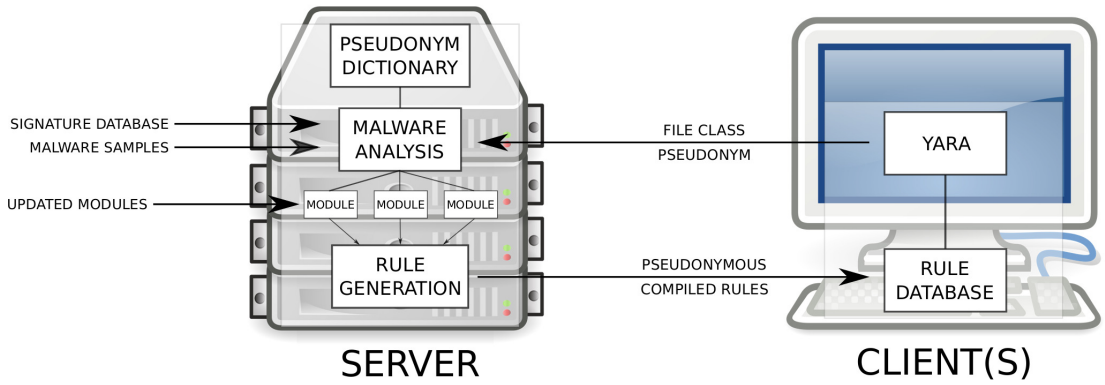

Fig. 1. The MASSE client-server architecture. The server modularly analyzes new malware samples, generates syntactic rules to detect them, and updates the client's databases in a pseudonymous and compiled form. The clients run a detection system based on YARA.

whitelist of system files and relevant user files to guarantee that the generated rules don't detect any of them as malicious. Rule update deployment uses canary testing, deploying first to a subset of testing systems to check the impact of the update before deployment to the whole system. Finally, rules contain different layers of detection confidence, and the system reacts accordingly. For instance, while a file classified as malware by its hash will be immediately quarantined, one classified just by some suspicious features can be transmitted to the server for further analysis before blocking user access to it. Together, these techniques guarantee that the generated YARA rules do not interfere with the client's systems.

The challenge of an open source architecture is to prevent malware creators from reverse-engineering it to learn how to shield their malware against it. To hinder reverse-engineering, the malware analysis modules are actually kept and distributed as obfuscated and closed source components (while the API they use, the client, and the server are open source). Importantly, these modules are only kept on the server, so that compromising a client does not allow attackers to access them. Finally, the YARA rule databases are delivered to the clients in a compiled, pseudonymous format. Rule precompilation increases YARA's efficiency, while pseudonimity ensures that even if an attacker was able to reverse-engineer the classification of their malware they would not be sure of whether the malware has been correctly identified. The clients have no information about the malware analysis techniques implemented in the server; when a sample is identified, its pseudonym is returned to the server that decodes it and proceeds accordingly.

b) Modular Malware Analysis: To achieve modularity in both the malware analysis and rule generation process we must understand how different types of malware analysis techniques can contribute not only with a malware classification but with syntactic properties representing the information that the technique used to determine the classification. Techniques that work directly on static properties like n-gram frequency, metadata, binary header properties, import/export tables, and entropy fingerprinting [1] can directly translate the most relevant properties to fragments of YARA rules. Techniques based on machine learning like classifiers and clustering algorithms [7] can isolate the most relevant features used to determine the classification or clustering of the given sample and use them as fragments of YARA rules, also weighting them by the relevance that each feature. Dynamic analysis techniques can extract important values or strings that the malware uses at runtime [8], like decryption keys, specific URLs, or credentials, and write YARA rule fragments to detect them statically in the binary.

Many of these techniques require their own signature databases or trained classifiers to work. These can be updated by the community with new signatures, as it happens nowadays with YARA rules, and companies can sell advanced analysis modules and subscription to updates of the modules and modules' databases as part of their business models.

References

[1] Mansour Ahmadi, Dmitry Ulyanov, Stanislav Semenov, Mikhail Trofimov, and Giorgio Giacinto. Novel feature extraction, selection and fusion for effective malware family classification. In Elisa Bertino, Ravi Sandhu, and Alexander Pretschner, editors, *Proceedings of the Sixth ACM on Conference on Data and Application Security and Privacy, CODASPY 2016, New Orleans, LA, USA, March 9-11, 2016*, pages 183–194. ACM, 2016.

[2] Victor Manuel Álvarez. *YARA*. VirusTotal, August 2016. http://virustotal.github.io/yara/.

[3] Stefan Bühlmann. *Introduction to YARA Rule Generator*. JoeSecurity, February 2015. http://blog.joesecurity.org/2015/02/introduction-yara-rule-generator.html.

[4] Chris Clark. *YaraGenerator*. XenoSec, August 2013. https://yaragenerator.com/.

[5] DeepEndResearch. *YARA Signature Exchange Group*, July 2017. http://www.deependresearch.org/2012/08/yara-signature-exchange-google-group.html.

[6] R. Fedler, M. Kulicke, and J. Schtte. An antivirus api for android malware recognition. In *2013 8th International Conference on Malicious and Unwanted Software: "The Americas" (MALWARE)*, pages 77–84, Oct 2013.

[7] Z.A. Markel. *Machine Learning Based Malware Detection*. Trident Scholar project report. 2015.

[8] Michael Sikorski and Andrew Honig. *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. No Starch Press, San Francisco, CA, USA, 1st edition, 2012.